Developing a Simple Web Browser in C#

1. Introduction

The purpose of this project is to design and implement a simple text-based web browser using **C#** and **Windows Forms (WinForms)** under **.NET 6**, running on Windows 11. The main objective is to provide basic HTTP communication between the client and remote web servers, display raw HTML code, and manage user navigation history, bookmarks, and homepage settings.

Assumptions made during development include:

- The browser will run in a Windows 11 environment with .NET Runtime 6 or higher.
- HTML rendering and JavaScript execution are **not** implemented, as per project rules.
- HTTP communications rely on the native **HttpClient** class from .NET libraries.
- File-based persistence is sufficient to maintain settings and user data (home page, bookmarks, history).

The intended outcome is a reliable, educational prototype demonstrating fundamental browser mechanisms at the HTTP level.

2. Requirements Checklist

Requirement	Description	Status
HTTP Request/Response	Send and receive HTTP messages using HttpClient	Fully Implemented
Display Raw HTML	Show raw HTML in text box	Fully Implemented
Display HTTP Status	Show response code (200, 400, 403, 404)	Fully Implemented
Page Reload	Allows reloading of current page	Fully Implemented
Home Page	Editable home URL, loads at startup	Fully Implemented
Bookmarks	Save, edit, delete bookmarks with names	Fully Implemented
History	Track URL requests with duplicate elimination	Fully Implemented
History Navigation	Backward and forward navigation	Fully Implemented
URL Extraction	Display first five `<a>` links	Fully Implemented
GUI	Menu, buttons, and panels built in WinForms	Fully Implemented
Persistence	Local text/JSON file storage	Fully Implemented
Rendering Engine	Display of graphical web pages	Not Implemented (beyond scope)

3. Design Considerations

Class Design:
The application adopts a modular class-based structure:

- `MainForm`: Handles GUI events, user interactions, and control binding.
- `HttpHandler`: Encapsulates HTTP operations using `HttpClient`.
- `DataManager`: Manages persistent data for history, bookmarks, and home URL.

Data Structures:

- Lists (`List<string>`) store history and fetched links.
- Dictionaries (`Dictionary<string,string>`) manage bookmarks paired with names.
- Plain text files are used for persistence.

GUI Design:
A **WinForms** interface provides buttons for navigation (Go, Back, Forward, Home), list boxes for bookmarks and history, a text box for URL entry, and a `RichTextBox` for displaying HTML code.

Advanced Functionalities:

- Automatic harvesting and listing of five hyperlink URLs.
- Deduplication logic in browsing history for improved user experience.
- Persistent home page settings read on program startup.

Performance Considerations:
`async` and `await` keywords ensure non-blocking network calls using asynchronous programming patterns inherent to C# and HttpClient.

4. Developer Guide

Code Organization:

- `Program.cs`: Initializes GUI and launches the main window.
- `MainForm.cs`: Core class handling events (e.g., `btnGo_Click`, `btnBack_Click`) and rendering results in controls.
- Persistent data stored in separate text files (`bookmarks.txt`, `history.txt`, and `home.txt`).

HTTP Communication:

- The **System.Net.Http.HttpClient** class executes all HTTP GET requests asynchronously.
- Response objects expose `StatusCode`, `ReasonPhrase`, and content read via `ReadAsStringAsync()`.

Persistent Data Storage:

- Bookmarks and history are stored in plain text for transparency and simplicity.
- Files are automatically read and written using `File.ReadAllText()` and `File.WriteAllLines()`.

GUI Technology:

- Developed entirely in **Windows Forms**, leveraging controls from the `System.Windows.Forms` namespace.
- The form includes `MenuStrip`, `RichTextBox`, and `ListBox` for accessible navigation and content management.
- Event-driven design ensures straightforward extensibility for future interface improvements.

Developer Hints for Extension:

- Implement HTML parsing using *HtmlAgilityPack* for link extraction.
- Replace file-based storage with a local SQLite database for relational persistence.
- Add a rendering toggle between raw HTML and a simple rendered view via a WebView control (optional).

5. Reflections on Language and Implementation

The C# language, particularly when used with **Visual Studio 2022** and **.NET 6**, provided excellent debugging capabilities and a well-integrated framework for GUI development. The **async/await** model greatly simplified network request management.

Key strengths identified:

- Strong type safety and error handling
- Well-documented .NET libraries
- Easy GUI prototyping in Visual Studio

Limitations observed:

- WinForms design is not responsive for dynamic resizing.
- Manual file-based storage lacks transactional safety.
- The current solution does not include SSL verification enhancement or caching.

Future improvements could include WPF-based GUI for modern UX, encrypted bookmark storage, and multitabbing functionality.

6. Conclusions

The delivered web browser successfully demonstrates the ability to handle HTTP requests, display responses, and preserve user data persistently. The modular structure and use of standard .NET libraries ensure compatibility with future versions of the platform.

Notable achievements include seamless asynchronous communications and a clear, functional GUI layout.
With more time, graphical HTML rendering and session management could be integrated to elevate this prototype toward a production-level educational browser.